Gerenciamento de memória

APRESENTAÇÃO

Em qualquer tarefa realizada no computador, diversos programas estão em execução apoiando no desenvolvimento, alguns de modo perceptível, como um editor de texto, outros de modo mais discreto, como o corretor ortográfico e o sistema operacional.

Os programas, para serem executados, precisam estar armazenados na memória do computador. Contudo, a tecnologia atual não alcançou o desenvolvimento de uma memória com armazenamento infinito, rápida o suficiente e com baixo custo. Para contornar essa limitação, os computadores são compostos por um conjunto de memórias, cada uma com um propósito específico.

Nesta Unidade de Aprendizagem, você vai aprender como funciona o gerenciamento de memória em um computador, observando as diferentes categorias de memórias usadas, vai verificar como os programas são endereçados e aprender as principais características de cada estratégia.

Bons estudos.

Ao final desta Unidade de Aprendizagem, você deve apresentar os seguintes aprendizados:

- Definir a hierarquia de memória de um computador.
- Descrever o espaço de endereçamento e de troca de memória.
- Identificar os modos de gerenciamento de memória e suas características.

DESAFIO

Os programas de usuário utilizados para diferentes tarefas ficam armazenados na memória do computador. Diferentemente do que a maioria das pessoas pensa, o computador não tem uma única memória, e sim uma combinação de diferentes tipos de memórias. Isso se dá pelas características de cada uma quanto a preço, volatilidade e velocidade.

Você trabalha em uma empresa de hospedagem de *sites*, servidores e serviços. A empresa oferece diversos planos mensais, que variam conforme a capacidade de processamento, transferência de dados, quantidade de memória RAM e espaço de armazenamento.

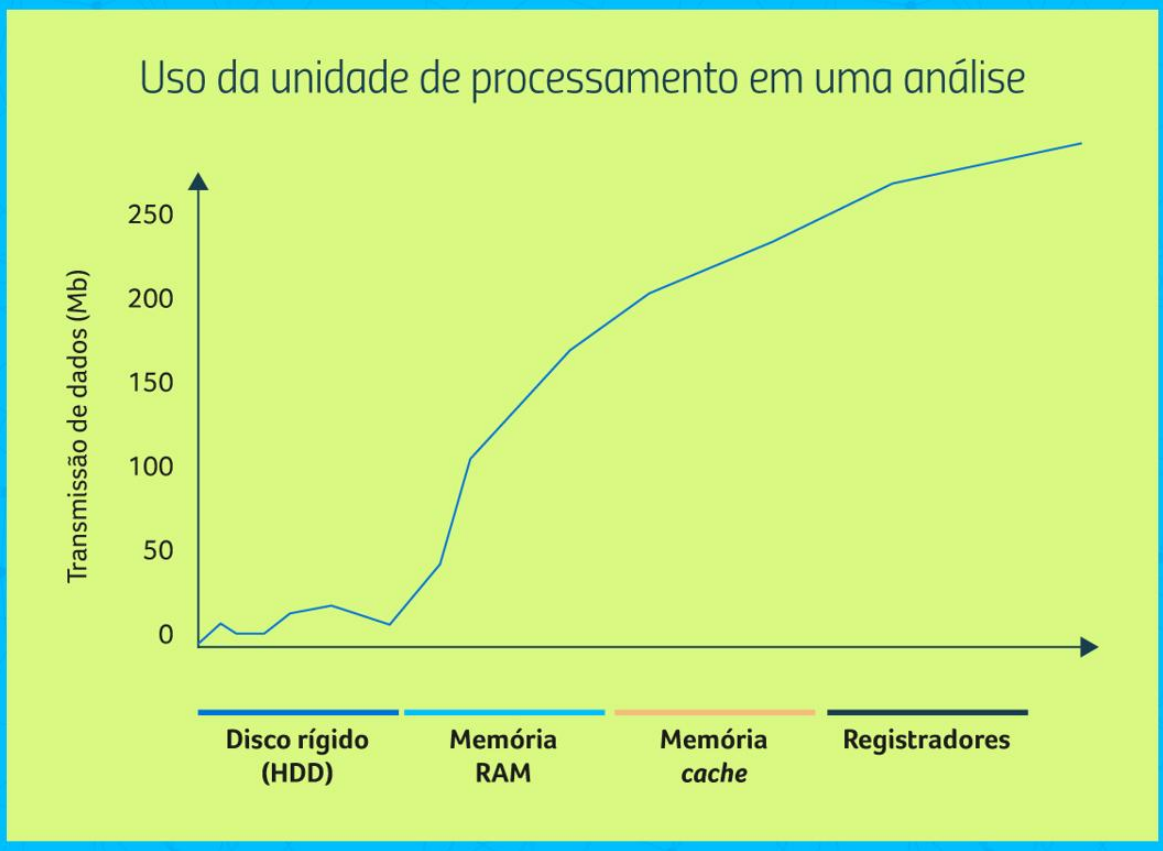
Um cliente da empresa entrou em contato reclamando da velocidade em que os dados estavam sendo armazenados no servidor contratado.

RECLAMAÇÃO

Os dados estão sendo transmitidos a uma velocidade muito menor que a velocidade da Internet que tenho, e contratei um dos melhores processadores do servidor disponível nos planos oferecidos.

ANALISANDO A TAXA DE TRANSMISSÃO

Após a reclamação, você foi analisar a taxa de transmissão de dados entre as memórias presentes no servidor do cliente e se deparou com o seguinte cenário:



INFOGRÁFICO

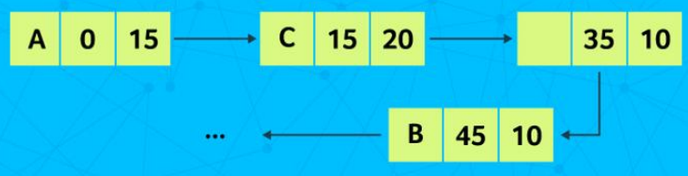
Na memória do computador, os processos são alocados e desalocados constantemente, muitos destes com diferentes tamanhos. Nesse processo de alocação, alguns espaços vazios podem surgir entre as regiões de memória alocadas, o que pode exigir que eles sejam gerenciados.

Confira, no Infográfico a seguir, o processo de gerenciamento de espaços vazios com listas encadeadas e alguns dos principais algoritmos.

ALGORITMOS DE GERENCIAMENTO DE ESPAÇOS VAZIOS

No gerenciamento de memória baseado em listas, a memória é dividida em segmentos de tamanhos diferentes que podem ser alocados e redimensionados conforme a necessidade, gerando **espaços vazios**.

Nesse caso, ao alocar um processo que ocupe 20 *bytes* no primeiro espaço vazio de 30 *bytes*, o processo será alocado. Contudo, ocorrerá um **novo espaço vazio**.



Entre alocações e desalocações de processos, muitos espaços vazios podem ser criados. É importante adotar uma boa estratégia de seleção de espaços vazios por meio de um algoritmo adequado, de modo a otimizar o processo de alocação e minimizar o desperdício de memória.

ALGORITMOS DE ALOCAÇÃO

FIRST FIT

O processo é alocado no primeiro segmento, a partir do início da lista, com tamanho suficiente para o processo.

NEXT FIT

O processo é alocado no primeiro segmento, a partir do último segmento alocado (que é memorizado na alocação anterior), com tamanho suficiente para o processo.

BEST FIT

O processo é alocado no segmento cujo tamanho seja o mais próximo possível do tamanho do processo (igual ou um pouco maior), na tentativa de desperdiçar menos espaço na alocação.

WORST FIT

Em oposição ao algoritmo anterior, o processo é alocado no segmento que tenha mais sobra de memória (o máximo possível), na tentativa de desperdiçar memória.

QUICK FIT

Os segmentos são organizados em listas de tamanhos mais solicitados, de modo a tentar agilizar a alocação da maioria dos processos.

CONTEÚDO DO LIVRO

Um computador é composto por diferentes tipos de memória, cada uma com seu propósito. As memórias secundárias, por exemplo, sejam elas discos rígidos, sejam elas discos de estado sólido, têm a função de armazenar permanentemente os dados. Por sua vez, para a execução, os programas utilizam a memória principal.

Durante a execução de um programa, os dados são transferidos entre as memórias conforme cada tipo de instrução. Assim, ao realizar uma operação matemática, os operandos precisaram ser direcionados até a unidade aritmética dentro do processador para o cálculo. Para isso acontecer, os valores das variáveis correspondentes aos operandos são transferidos para a memória *cache* e, posteriormente, alocados em registradores no processador. Após o processamento do cálculo, o caminho precisará ser feito para o resultado da operação. Todo esse processo requer o gerenciamento dos diferentes tipos de memória do computador.

No trecho da obra *Sistemas operacionais*, você irá compreender como ocorre o gerenciamento de memória dos computadores, além de verificar a hierarquia de memória, como o endereçamento é feito e as diferentes estratégias de alocação de memória.

Boa leitura.

O Livro do MINIX

SISTEMAS OPERACIONAIS

Projeto e Implementação

TERCEIRA EDIÇÃO

INCLUI CD-ROM

Andrew S. Tanenbaum
Albert S. Woodhull

Bookman®

T164s Tanenbaum, Andrewa S.
Sistemas operacionais [recurso eletrônico] : projeto e implementação / Andrew S. Tanenbaum, Albert S. Woodhull ; tradução João Tortello. – 3. ed. – Dados eletrônicos. – Porto Alegre : Bookman, 2008.

Editado também como livro impresso em 2008.
ISBN 978-85-7780-285-2

1. Sistemas operacionais. I. Woodhull, Albert S. II. Título.

CDU 004.4

Catalogação na publicação: Mônica Ballejo Canto – CRB 10/1023

4.1 GERENCIAMENTO BÁSICO DE MEMÓRIA

Os sistemas de gerenciamento de memória podem ser divididos em duas classes fundamentais: aqueles que alternam os processos entre a memória principal e o disco durante a execução (*swapping*) e aqueles que não alternam. Estes últimos são mais simples, portanto, estudaremos primeiro. Posteriormente, neste capítulo, examinaremos o *swapping* e a paginação. Ao longo de todo este capítulo o leitor deverá lembrar que *swapping* e paginação são artifícios usados para contornar a falta de memória principal suficiente para conter todos os programas e dados simultaneamente. Se a memória principal ficar tão grande que haja realmente o suficiente, os argumentos a favor de um tipo de esquema de gerenciamento de memória ou de outro podem se tornar obsoletos.

Por outro lado, conforme mencionamos anteriormente, o software parece crescer tão rápido quanto a memória; portanto, o gerenciamento eficiente da memória sempre pode ser necessário. Nos anos 80, havia muitas universidades que usavam um sistema de compartilhamento de tempo com dezenas de usuários (mais ou menos satisfeitos), em um VAX de 4 MB. Agora, a Microsoft recomenda ter pelo menos 128 MB para um sistema Windows XP monousuário. A tendência em direção à multimídia impõe ainda mais exigências sobre a memória; portanto, um bom gerenciamento de memória provavelmente ainda vai ser necessário no mínimo por mais uma década.

4.1.1 Monoprogramação sem *swapping* ou paginação

O esquema de gerenciamento de memória mais simples possível é executar apenas um programa por vez, compartilhando a memória entre esse programa e o sistema operacional. Três variações sobre esse tema aparecem na Figura 4-1. O sistema operacional pode estar na parte inferior da memória na RAM (*Random Access Memory* – memória de acesso aleatório), como se vê na Figura 4-1(a), pode estar na ROM (*Read-Only Memory* – memória somente de leitura), na parte superior da memória, como se vê na Figura 4-1(b), ou os *drivers* de dispositivo podem estar na parte superior da memória em uma ROM e o restante do sistema na RAM abaixo dela, como se vê na Figura 4-1(c). O primeiro modelo foi usado inicialmente em computadores de grande porte e em minicomputadores, mas hoje em dia raramente é usado. O segundo modelo é usado em alguns *palmtops* e em sistemas embarcados. O terceiro modelo foi usado pelos primeiros computadores pessoais (por exemplo, executando MS-DOS), onde a parte do sistema que fica na ROM é chamada de **BIOS** (*Basic Input Output System* – sistema básico de entrada e saída).

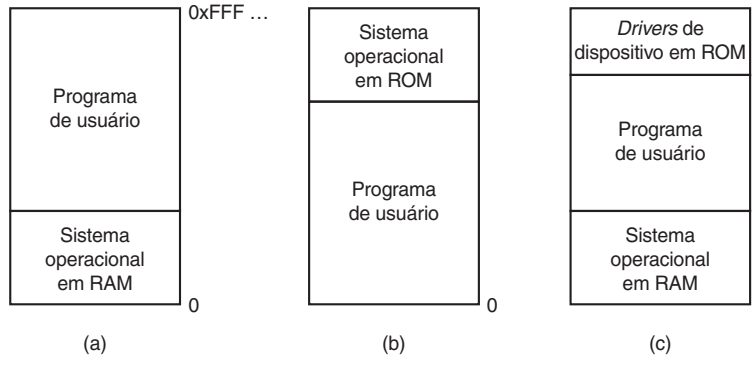


Figura 4-1 Três maneiras simples de organizar a memória com um sistema operacional e um único processo de usuário. Também existem outras possibilidades.

Quando o sistema é organizado dessa maneira, apenas um processo por vez pode estar em execução. Assim que o usuário digita um comando, o sistema operacional copia o programa solicitado do disco para a memória e o executa. Quando o processo termina, o sistema operacional exibe um caractere de aviso e espera por um novo comando. Ao receber o comando, ele carrega um novo programa na memória, sobrescrevendo o primeiro.

4.1.2 Multiprogramação com partições fixas

A não ser em sistemas embarcados muito simples, a monoprogramação dificilmente é usada hoje em dia. A maioria dos sistemas modernos permite que vários processos sejam executados ao mesmo tempo. Ter vários processos executando simultaneamente significa que, quando um processo está bloqueado esperando o término de uma operação de E/S, outro processo pode usar a CPU. Assim, a multiprogramação aumenta a utilização da CPU. Os servidores de rede sempre têm a capacidade de executar vários processos (para diferentes clientes) ao mesmo tempo, mas hoje em dia a maioria das máquinas clientes (isto é, de *desktop*) também tem essa capacidade.

A maneira mais fácil de obter multiprogramação é simplesmente dividir a memória em até *n* partições (possivelmente de tamanhos diferentes). Esse particionamento pode ser feito manualmente, por exemplo, quando o sistema é inicializado.

Quando chega um *job*, ele pode ser colocado na fila de entrada da menor partição grande o bastante para contê-lo. Como as partições são fixas nesse esquema, todo espaço não utilizado por um *job* em uma partição é desperdiçado, enquanto esse *job* é executado. Na Figura 4-2(a), vemos como é esse sistema de partições fixas e filas de entrada separadas.

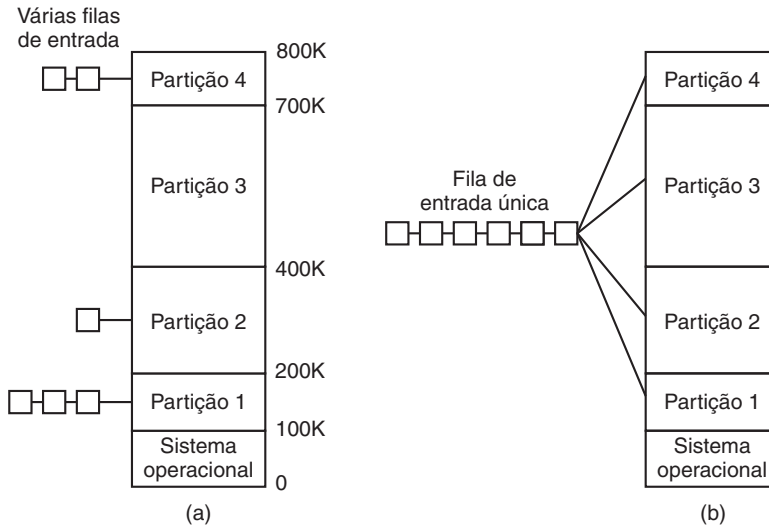


Figura 4-2 (a) Partições fixas de memória com filas de entrada separadas para cada partição. (b) Partições fixas de memória com uma única fila de entrada.

A desvantagem de ordenar os *jobs* recebidos em filas separadas se torna evidente quando a fila de uma partição grande está vazia, mas a de uma partição pequena está cheia, como acontece nas partições 1 e 3 da Figura 4-2(a). Aqui, os *jobs* pequenos têm de esperar para entrar na memória, mesmo havendo muita memória livre. Uma organização alternativa é manter uma única fila, como na Figura 4-2(b). Quando uma partição fica livre, o *job* mais próximo do

início da fila, e que caiba na partição vazia, poderia ser carregado nessa partição e executado. Como é indesejável desperdiçar uma partição grande com um *job* pequeno, uma estratégia diferente é, quando uma partição ficar livre, pesquisar a fila de entrada inteira e escolher o maior *job* que caiba nela. Note que este último algoritmo discrimina os *jobs* pequenos, tratando-os como indignos de terem uma partição inteira, embora normalmente seja desejável fornecer o melhor serviço para *jobs* menores (freqüentemente *jobs* interativos) e não o pior.

Uma saída é dispor de pelo menos uma partição pequena. Essa partição permitirá que os *jobs* pequenos sejam executados sem precisar alocar uma partição grande para eles.

Outra estratégia é ter uma regra dizendo que um *job* pronto para executar não pode ser preterido mais do que *k* vezes. Sempre que for preterido, ele recebe um ponto. Quando tiver adquirido *k* pontos, ele não poderá ser preterido novamente.

Esse sistema, com partições fixas configuradas de manhã pelo operador e não mais alteradas depois disso, foi usado durante muitos anos pelo OS/360 em computadores IBM de grande porte. Ele se chamava **MFT** (multiprogramação com um número fixo de tarefas ou OS/MFT). Ele é simples de entender e igualmente simples de implementar: os *jobs* (tarefas) recebidos são enfileirados até que uma partição conveniente esteja disponível, no momento em que o *job* é carregado nessa partição e executado até terminar. Entretanto, hoje em dia poucos (se houver) sistemas operacionais suportam esse modelo, mesmo em sistemas de lote de computadores de grande porte.

4.1.3 Realocação e proteção

A multiprogramação introduz dois problemas básicos que devem ser resolvidos: realocação e proteção. Veja a Figura 4-2. A partir da figura fica claro que diferentes tarefas serão executadas em diferentes endereços. Quando um programa é ligado (isto é, o programa principal, as funções escritas pelo usuário e as funções de biblioteca são combinados em um único espaço de endereçamento), o ligador deve saber em que endereço o programa começará na memória.

Por exemplo, suponha que a primeira instrução seja uma chamada para uma função no endereço absoluto 100, dentro do arquivo binário produzido pelo ligador. Se esse programa for carregado na partição 1 (no endereço 100K), essa instrução pulará para o endereço absoluto 100, que está dentro do sistema operacional. O que é preciso é uma chamada para 100K + 100. Se o programa for carregado na partição 2, ele deverá ser executado como uma chamada para 200K + 100 e assim por diante. Esse problema é conhecido como problema da **realocação**.

Uma possível solução é modificar realmente as instruções quando o programa é carregado na memória. Os programas carregados na partição 1 têm 100K somados a cada endereço, os programas carregados na partição 2 têm 200K somados aos endereços e assim por diante. Para realizar a realocação dessa forma, durante a carga, o ligador precisa incluir no programa binário uma lista, ou um mapa de bits, informando quais palavras do programa são endereços a serem corrigidas (realocadas) e quais são códigos de operação, constantes ou outros itens que não devem ser realocadas. O OS/MFT funcionava assim.

A realocação durante a carga não resolve o problema da proteção. Um programa maldoso sempre pode construir uma nova instrução e pular para ela. Como os programas nesse sistema usam endereços de memória absolutos em vez de endereços relativos ao valor de um registrador, não há como impedir que um programa construa uma instrução que leia ou escreva qualquer palavra na memória. Nos sistemas multiusuário, é altamente indesejável permitir que os processos leiam e escrevam na memória pertencente a outros usuários.

A solução escolhida pela IBM para proteger o 360 foi dividir a memória em blocos de 2 Kbytes e atribuir um código de proteção de 4 bits a cada bloco. O PSW (*Program Status*

Word) continha uma chave de 4 bits. O hardware do 360 detectava qualquer tentativa por parte de um processo em execução de acessar memória cujo código de proteção fosse diferente da chave PSW. Como apenas o sistema operacional podia mudar os códigos de proteção e a chave, os processos de usuário eram impedidos de interferir uns nos outros e no sistema operacional em si.

Uma solução alternativa para os problemas de realocação e proteção é fornecer dois registradores de hardware especiais, chamados de **base** e **limite**. Quando um processo é escalonado, o registrador de base é carregado com o endereço do início de sua partição e o registrador de limite é carregado com o tamanho dessa partição. Todo endereço de memória gerado tem o conteúdo do registrador de base automaticamente somado a ele, antes de ser enviado para a memória. Assim, se o registrador de base contém o valor 100K, uma instrução CALL 100 é efetivamente transformada em uma instrução CALL 100K + 100, sem que a instrução em si seja modificada. Os endereços também são verificados em relação ao registrador de limite para garantir que não tentem endereçar memória fora da partição corrente. O hardware protege os registradores de base e de limite para impedir que programas de usuário os modifiquem.

Uma desvantagem desse esquema é a necessidade de efetuar uma adição e uma comparação em cada referência de memória. As comparações podem ser feitas rapidamente, mas as adições são lentas, devido ao tempo de propagação do transporte, a não ser que sejam usados circuitos de adição especiais.

O CDC 6600 – o primeiro supercomputador do mundo – usava esse esquema. A CPU Intel 8088 usada pelo IBM PC original utilizava uma versão ligeiramente menos eficiente desse esquema – com registradores de base, mas sem registradores de limite. Atualmente, poucos computadores o utilizam.

4.2 *SWAPPING*

Com um sistema de lotes, organizar a memória em partições fixas é simples e eficiente. Cada *job* é carregado em uma partição quando chega no começo da fila. O *job* permanece na memória até que tenha terminado. Contanto que *jobs* suficientes possam ser mantidos em memória para conservar a CPU ocupada o tempo todo, não há porque usar algo mais complicado.

Com sistemas de compartilhamento de tempo a situação é diferente. Às vezes, não há memória principal suficiente para conter todos os processos correntemente ativos, de modo que os processos excedentes devem ser mantidos no disco e trazidos para execução dinamicamente.

Podem ser usadas duas estratégias gerais de gerenciamento de memória, dependendo (em parte) do hardware disponível. A estratégia mais simples, chamada de ***swapping***, consiste em trazer cada processo em sua totalidade, executá-lo por algum tempo e, então, colocá-lo de volta no disco. A outra estratégia, chamada de **memória virtual**, permite que os programas sejam executados mesmo quando estão apenas parcialmente na memória principal. A seguir, estudaremos o *swapping*; na Seção 4.3, examinaremos a memória virtual.

O funcionamento de um sistema de *swapping* está ilustrado na Figura 4-3. Inicialmente, apenas o processo *A* está na memória. Então, os processos *B* e *C* são criados ou recuperados do disco. Na Figura 4-3(d), *A* é enviado para o disco. Então, *D* entra e *B* sai. Finalmente, *A* entra outra vez. Como *A* está agora em um local diferente, os endereços contidos nele devem ser realocados, ou pelo software, quando ele é colocado na memória, ou (mais provavelmente) pelo hardware, durante a execução do programa.

A principal diferença entre as partições fixas da Figura 4-2 e as partições variáveis da Figura 4-3 é que, nestas, o número, a posição e o tamanho das partições variam dinamica-

mente à medida que os processos entram e saem, ao passo que, nas primeiras, as partições são fixas. A flexibilidade de não estar vinculado a uma determinada partição, que pode ser grande ou pequena demais, melhora a utilização da memória, mas também complica a alocação e a liberação da memória, assim como seu monitoramento.

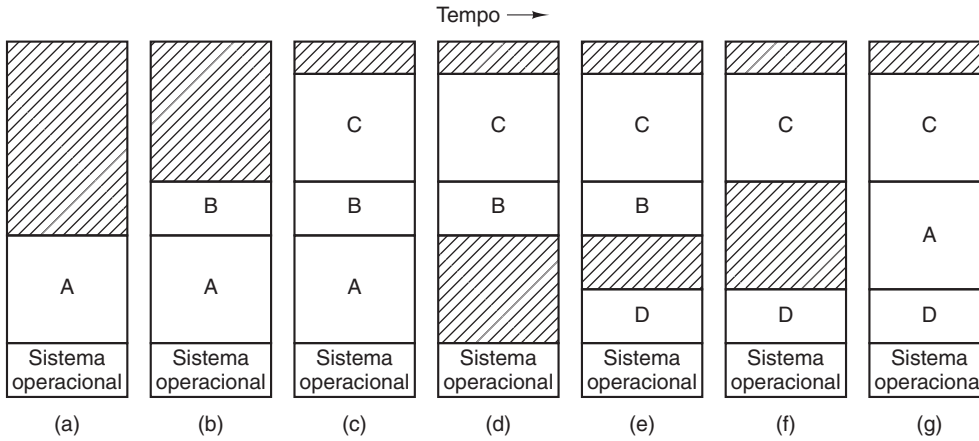


Figura 4-3 A alocação da memória muda à medida que os processos entram e saem da memória. As regiões sombreadas representam memória não utilizada.

Quando o *swapping* cria várias lacunas na memória, é possível combiná-las em apenas uma lacuna grande, movendo todos os processos o mais para baixo possível. Essa técnica é conhecida como **compactação de memória**. Normalmente ela não é feita porque exige muito tempo de CPU. Por exemplo, em uma máquina de 1 GB, que executa cópia com uma velocidade de 2 GB/s (0,5 ns/byte), demoraria cerca de 0,5 s para compactar toda a memória. Isso pode não parecer muito tempo, mas seria perceptível para um usuário que estivesse vendo um vídeo.

Um ponto que merece ser considerado é a quantidade de memória que deve ser alocada para um processo quando ele é criado ou recuperado do disco. Se os processos são criados com um tamanho fixo que nunca muda, então a alocação é simples: o sistema operacional aloca exatamente o que é necessário, nem mais nem menos.

Entretanto, se os segmentos de dados dos processos podem crescer, por exemplo, pela alocação dinâmica de memória a partir de um *heap*, como acontece em muitas linguagens de programação, ocorre um problema quando um processo tentar crescer. Se houver uma lacuna adjacente ao processo, ela poderá ser alocada e o processo poderá crescer utilizando a lacuna. Por outro lado, se o processo for adjacente a outro processo, o processo em crescimento terá de ser movido para uma lacuna na memória que seja grande o suficiente para ele ou, então, um ou mais processos terão de ser enviados para o disco para criar uma lacuna suficientemente grande. Se um processo não puder crescer na memória e a área de *swap* no disco estiver cheia, o processo terá de esperar ou ser eliminado.

Se a expectativa for de que a maioria dos processos crescerá quando executados, provavelmente será uma boa idéia alocar um pouco de memória extra, quando um processo for colocado ou movido da memória, para reduzir a sobrecarga associada à movimentação ou *swapping* de processos que não cabem mais em sua memória alocada. Entretanto, ao fazer *swapping* dos processos no disco, apenas a memória realmente em uso deverá ser transferida; é um desperdício transferir a memória extra. Na Figura 4-4(a), vemos uma configuração de memória na qual o espaço para crescimento foi alocado para dois processos.

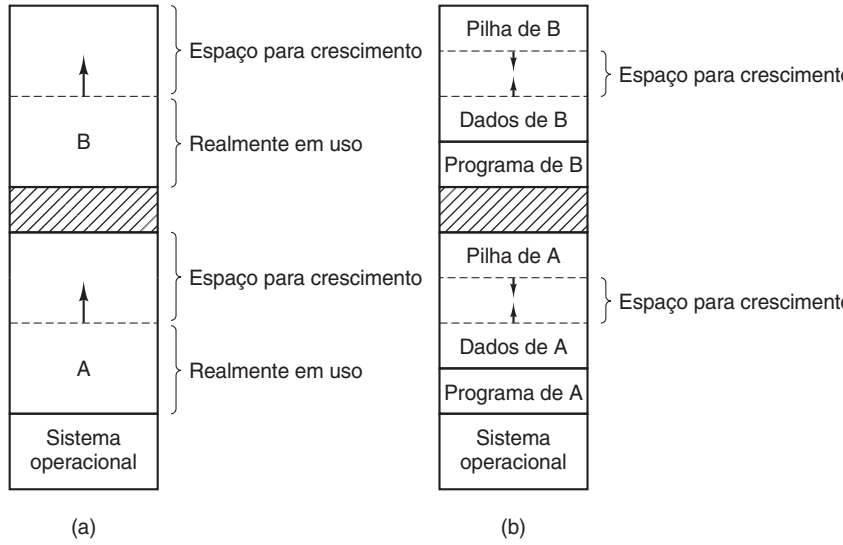


Figura 4-4 (a) Alocando espaço para um segmento de dados em crescimento. (b) Alocando espaço para uma pilha em crescimento e para um segmento de dados em crescimento.

Se os processos puderem ter dois segmentos em crescimento, por exemplo, o segmento de dados sendo usado como *heap* para variáveis alocadas e liberadas dinamicamente e um segmento de pilha para as variáveis locais e endereços de retorno, sugere-se uma organização alternativa, a saber, a da Figura 4-4(b). Nessa figura, vemos que cada processo ilustrado tem uma pilha, no início de sua memória alocada, que está crescendo para baixo, e um segmento de dados imediatamente após o texto do programa, que está crescendo para cima. A memória entre eles pode ser usada por qualquer um dos dois segmentos. Se ela acabar, um dos dois processos terá de ser movido para uma lacuna com espaço suficiente, sendo retirado da memória até que uma lacuna grande o bastante possa ser criada, ou ser eliminado.

4.2.1 Gerenciamento de memória com mapas de bits

Quando a memória é atribuída dinamicamente, o sistema operacional precisa gerenciá-la. Em termos gerais, há duas maneiras de monitorar a utilização da memória: mapas de bits e listas de regiões livres. Nesta seção e na próxima, veremos esses dois métodos.

Com um mapa de bits, a memória é dividida em unidades de alocação, talvez tão pequenas quanto algumas palavras e talvez tão grandes quanto vários quilobytes. Há um bit no mapa de bits, correspondendo a cada unidade de alocação, que é 0 se a unidade estiver livre e 1 se estiver ocupada (ou vice-versa). A Figura 4-5 mostra parte da memória e o mapa de bits correspondente.

O tamanho da unidade de alocação é uma importante questão de projeto. Quanto menor for a unidade de alocação, maior será o mapa de bits. Entretanto, mesmo com uma unidade de alocação tão pequena quanto 4 bytes, 32 bits de memória exigirão apenas 1 bit do mapa. Uma memória de 32n bits usará n bits do mapa; portanto, o mapa de bits ocupará apenas 1/33 da memória. Se for escolhida uma unidade de alocação grande, o mapa de bits será menor, mas uma quantidade de memória apreciável poderá ser desperdiçada na última unidade alocada ao processo, isso se o tamanho do processo não for um múltiplo exato da unidade de alocação.

Um mapa de bits proporciona uma maneira simples de monitorar palavras de memória em uma quantidade fixa de memória, pois o tamanho do mapa de bits depende apenas do

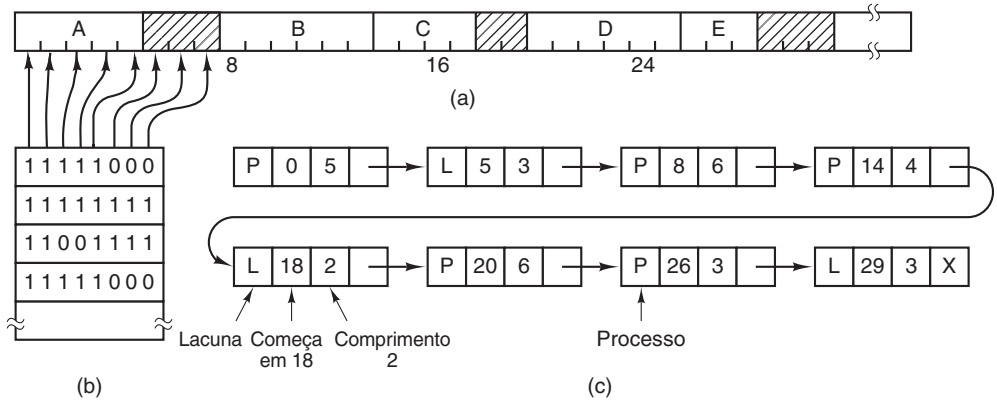


Figura 4-5 (a) Uma parte da memória com cinco processos e três lacunas. Os tracinhos mostram as unidades de alocação de memória. As regiões sombreadas (0 no mapa de bits) estão livres. (b) O mapa de bits correspondente. (c) As mesmas informações como uma lista.

tamanho da memória e da unidade de alocação. O principal problema disso é que, quando for decidido trazer um processo de *k* unidades para a memória, o gerenciador de memória deverá pesquisar o mapa de bits para encontrar uma seqüência de *k* bits em 0 consecutivos no mapa. Pesquisar um mapa de bits para encontrar uma seqüência de determinado comprimento é uma operação lenta (porque a seqüência pode se esparramar nos limites de palavra do mapa); esse é um argumento contra os mapas de bits.

4.2.2 Gerenciamento de memória com listas encadeadas

Outra maneira de monitorar a memória é manter uma lista encadeada de segmentos de memória alocados e livres, onde um segmento é um processo ou uma lacuna entre dois processos. A memória da Figura 4-5(a) está representada na Figura 4-5(c) como uma lista encadeada de segmentos. Cada entrada na lista especifica uma lacuna (L) ou processo (P), o endereço em que inicia, o comprimento e um ponteiro para a próxima entrada.

Nesse exemplo, a lista de segmentos está ordenada pelo endereço. Ordenar dessa maneira tem a vantagem de que, quando um processo termina, ou é enviado para o disco, atualizar a lista é simples. Normalmente, um processo que está terminando tem dois vizinhos (exceto quando está no início ou no final da memória) que podem ser processos ou lacunas, levando às quatro combinações mostradas na Figura 4-6. Na Figura 4-6(a), atualizar a lista exige substituir um P por um L. Na Figura 4-6(b) e também na Figura 4-6(c), duas entradas são aglutinadas em uma e a lista se torna uma única entrada mais curta. Na Figura 4-6(d), três entradas são aglutinadas e dois itens são removidos da lista. Como a entrada da tabela de

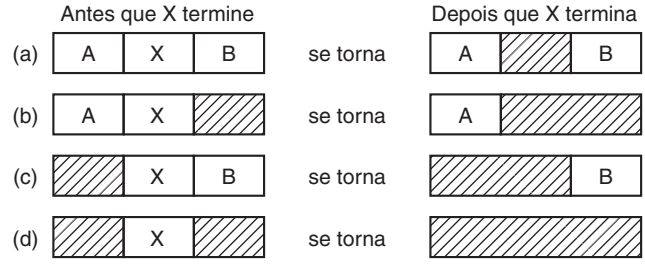


Figura 4-6 Quatro combinações de vizinhos para o processo que está terminando, *X*.

Encerra aqui o trecho do livro disponibilizado para esta Unidade de Aprendizagem. Na Biblioteca Virtual da Instituição, você encontra a obra na íntegra.

DICA DO PROFESSOR

Um fator importante para todo sistema operacional é o gerenciamento e o compartilhamento dos recursos. Isso não é diferente quanto ao uso da memória principal do computador, que deve ser distribuída entre o sistema operacional e os programas de usuário para garantir o correto funcionamento de todas as partes.

Mesmo que o objetivo possa ser simples, o sistema operacional, por meio do gerenciador de memória, precisa lidar com diferentes fatores que aumentam a complexidade do gerenciamento da memória, como, por exemplo, a implementação de multiprogramação.

Confira, na Dica do Professor, os diferentes modos ou categorias de gerenciamento de memória.

Conteúdo interativo disponível na plataforma de ensino!

EXERCÍCIOS

1) **A multiprogramação requer que dois ou mais programas estejam carregados na memória para a execução. Cada programa apresenta suas próprias variáveis referenciadas, os próprios endereços.**

   Um computador tem dois programas, A e B: o programa A tem uma variável F no endereço 28, e o programa B uma variável G no endereço. Se o programa A tiver o espaço de memória iniciado na posição 0 e o programa B na posição 200, quais serão as posições de endereço das variáveis F e G?

A) A variável F ficará no endereço -28, e a variável G, no endereço 172.

B) A variável F ficará no endereço 0, e a variável G, no endereço 200.

C) A variável F ficará no endereço 28, e a variável G, no endereço 228.

D) A variável F ficará no endereço 200, e a variável G, no endereço 0.

E) A variável G ficará no endereço 28, e a variável F, no endereço 228.

2) **A respeito do carregamento de processos na memória física do computador, existem dois métodos principais: a troca de processos e a memória virtual.**

Quanto ao primeiro método, a troca de processos, também conhecido como *swapping*, pode-se dizer:

A) Impossibilita a criação de espaços adicionais de memória ao processo.

B) Pode gerar espaços vazios entre os processos na memória.

C) Carrega os processos parcialmente.

D) Impede a alocação dos processos de modo contíguo.

E) Quando um processo cresce, os vizinhos são transferidos para outras áreas.

3) **Quando a memória é atribuída dinamicamente aos processos, o sistema operacional precisa controlar quais espaços estão ocupados e quais estão livres, para distribuí-los conforme a necessidade dos processos. Os principais métodos de gerenciamento são o mapa de *bits* e as listas encadeadas.**

Sobre esses métodos de controle, pode-se afirmar:

A) O gerenciamento de memória com mapa de *bits* divide-a em unidades com tamanhos variados; se a unidade estiver ocupada, representa-se o *bit* com 1, se disponível, com 0.

B) O gerenciamento de memória com listas encadeadas divide-a em segmentos de tamanho igual, que podem ser ocupados ou estar disponíveis para uso.

C) Um computador com memória física de 1 *megabyte* usa o método de mapa de *bits* com

unidades de tamanho de 4 *kilobytes;* logo, são necessários 512 *bits* para representar o mapa.

D) Um computador com memória física de 1 *megabyte* usa o método de mapa de *bits* com unidades de tamanho de 512 *bytes;* logo, serão necessários 2.048 *bits* para representar o mapa.

E) O gerenciamento de memória com listas encadeadas não permite que os segmentos de memória disponíveis e adjacentes sejam unidos formando um segmento maior.

4) No gerenciamento de memória com listas encadeadas, existem diferentes algoritmos para seleção do segmento para alocação dos processos.

Sobre esses algoritmos, é correto afirmar:

A) O algoritmo *next fit* sempre aloca o processo no segundo seguinte encontrado.

B) O algoritmo *best fit* aloca o processo no melhor segmento possível, ou seja, o segmento de maior espaço.

C) O algoritmo *worst fit* aloca o processo no menor segmento disponível.

D) O algoritmo *best fit* aloca o processo no segmento em que sobre o menor espaço possível.

E) O algoritmo *next fit* aloca o processo no segmento ao lado do segmento atual.

5) Na implementação do espaço de endereçamento de processos, algumas abordagens utilizam dois registradores especiais: o registrador-base e o registrador-limite.

Quanto a esses registradores, pode-se afirmar:

A) O registrador-base define o início do espaço de endereçamento.

B) O registrador-limite define o tamanho do espaço de endereçamento em *bytes*.

C) O registrado-base define o fim do espaço de endereçamento.

D) O registrador-limite define o início do espaço de endereçamento.

E) Dois processos podem conter o mesmo registrador-base, mas não o limite.

NA PRÁTICA

O processo de gerenciamento de memória é responsável por alocar as regiões de memória aos seus respectivos processos, permitindo a execução dos processos e o carregamento dos dados necessários. Quando o sistema oferece suporte à multiprogramação, a tarefa se torna mais complexa, pois é necessário gerenciar o compartilhamento da memória por todos os processos, **controlar o acesso ao espaço reservado para cada um** e garantir que um processo não corrompa a memória do outro.

Confira, Na Prática, como Lucas e sua equipe atuaram no gerenciamanento de memória em sistemas embarcados.

GERENCIAMENTO DE MEMÓRIA EM SISTEMAS EMBARCADOS

Lucas trabalha como desenvolvedor em um projeto de aprimoramento de um sistema de computador de bordo de carros.

Atualmente, o sistema monitora informações relativas ao funcionamento do carro, e planeja-se adicionar funcionalidades de:

Multimídia — Conectividade com a Internet — e GPS...

...além de alguns sensores.

O funcionamento dos novos *softwares* fará uso da memória do computador de bordo quando o usuário acioná-lo e o mantiver em execução. Ao término, a memória em uso é liberada para os outros programas.

Assim, será possível a **execução de todos os programas sem necessidade de memória adicional**.

Em geral, além do programa de monitoramento do funcionamento do carro, um ou dois programas estariam ativos simultaneamente.

O PROBLEMA

Após implementarem os programas, Lucas e sua equipe foram testá-los em um computador de bordo disponível. No entanto, em todos os testes, aconteceram falhas na execução.

- A parte estranha é que nos testes prévios os programas funcionaram bem nos computadores utilizados no desenvolvimento; contudo, no computador de bordo, isso não aconteceu.
- Os erros apresentados indicavam que as instruções não foram executadas corretamente.

INVESTIGAÇÃO

Registrador-base

Como solução, Lucas propôs que a cada execução de um programa fosse criado um par de registrados, uma base e um limite, os quais delimitariam a região de memória do programa.

Acessando o endereço 50 do programa B

Assim, quando esse programa tentar acessar uma posição da memória, a posição seria acrescida com o registrador-base, levando diretamente ao endereço correto.

Registrador-limite

Além disso, com a definição dos limites, os programas não acessam as memórias uns dos outros.

CONCLUSÃO

A partir da solução proposta, os programas implementados funcionaram perfeitamente, sem qualquer erro ou conflito no usuário da memória ou na definição do espaço de **endereçamento por meio de registradores**.

SAIBA +

Para ampliar o seu conhecimento a respeito desse assunto, veja abaixo as sugestões do professor:

Introdução ao gerenciamento de memória

Nesta videoaula da Univesp TV, você pode rever os conceitos relacionados ao gerenciamento de memória.

Conteúdo interativo disponível na plataforma de ensino!

Como funcionam as diferentes memórias quando um computador está em uso

Compreenda o funcionamento dos diferentes tipos de memória presentes em computador e o propósito de cada um.

Conteúdo interativo disponível na plataforma de ensino!

Gerência de memória

Conheça, de forma ilustrativa, o processo de gerência de memória que o sistema operacional realiza.

Conteúdo interativo disponível na plataforma de ensino!